

UNITED STATES PATENT APPLICATION

For

A METHOD FOR DYNAMICALLY DESIGNATING INITIALIZATION
MODULES AS RECOVERY CODE

INVENTORS:

VINCENT ZIMMER

JOHN LAMBINO

ANDREW FISH

SUSIE LI

SHAM DATTA

WILLIAM STEVENS

Prepared By:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026

(408) 720-8300

"Express Mail" mailing label number: EL 867 652 819 US

Date of Deposit: August 30, 2001

I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" under 37 C.F.R. § 1.10 on the date indicated above and that this paper or fee has been addressed to the Assistant Commissioner for Patents, Washington, D. C. 20231

Maureen R. Pettibone

(Typed or printed name of person mailing paper or fee)

Maureen R. Pettibone

(Signature of person mailing paper or fee)

8/30/01

(Date signed)

4065460

A METHOD FOR DYNAMICALLY DESIGNATING INITIALIZATION MODULES AS RECOVERY CODE

FIELD OF THE INVENTION

[0001] This invention relates generally to the basic input/output system (BIOS) of a computer system, and more specifically to the designation of recovery code in a computer system having a component firmware architecture.

BACKGROUND OF THE INVENTION

[0002] Recovery code is a typically small subset of the BIOS code that allows the system access through an input/output (I/O) device to provide minimal initialization in the event of corruption (e.g., hardware failure, security error, etc.) to effect the restoration of the entire firmware image. That is, the system is able to boot to a point so that a copy of lost data can be read from a specified peripheral. Typically what a BIOS vendor does is to designate a portion of the BIOS firmware as recovery and protect that code. The rest of the BIOS would be designated as non-recovery and would not be protected. The recovery/non-recovery designation is static. Therefore, in the event of a bug fix or a new feature the entire BIOS must be changed, or at least the entire recovery block.

[0003] A recent development in computer system firmware is an extensible firmware interface framework that allows software vendors to develop operating systems programs that can be used with a variety of central processing units (CPUs).

[0004] Componentization of the BIOS presents the issue of evolving recovery code. In a component-based firmware file system, one or more firmware volumes may contain recovery initialization modules (RIMs). Initially the system vendor may aggregate initialization modules from multiple parties and designate those initialization

modules necessary for recovery. The RIMs are kept in a fault-tolerant block, the size of which is determined by the system manufacturer. Typically, this fault-tolerant block may be 64 kilobytes in size. The number of initialization modules designated as RIMs is typically kept to a minimum because RIMs must be rendered fault tolerant, which is expensive. During the life of the system, the consumer may wish to update the firmware by replacing/adding initialization modules. These modules may not have been designated as a RIM at the time of design, but a subset of the RIMs may depend on the module. Therefore, if corruption occurs, a full recovery may not be possible. For example, the added initialization module may be responsible for initializing memory. Since the added module was not rendered fault tolerant by inclusion in the recovery set, the system may not have the necessary component to initialize the memory controller.

[0005] An exemplary recovery procedure for a computing system having component firmware architecture is described as follows. The dispatcher is notified to transition to recovery mode by the setting of a particular bit, i.e. a “” bit. The dispatcher clears out the fault tolerant by inclusion in the recovery set, the system may not have the necessary component to initialize the memory controller. An exemplary recovery list of initiation modules dispatched and sets the pass state to Pass 1. The dispatcher then sets the Boot_in_Recovery_Mask. Typically, a reset is issued at this point to reset hardware back to a pristine state. The current boot path needs to be saved in an area that is unaffected by a reset and restored after the reset. The dispatcher then restarts the dispatch in recovery mode. IMs that require different actions in “Recovery” mode than normal mode and IMs known to be required for “Recovery” mode need to be designated as such. For example by setting a recovery bit in the IMs FFS header. These modules along with the security startup code and the initialization core make up the recovery block.

[0006] The dispatcher commences searching for modules to invoke and only invokes those modules that are actually marked as being for “recovery dispatch” in the course of its search. These modules might also serve dual purpose for the normal boot, such as having just one IM for memory initialization, but when these modules detect that the boot in recovery bit is set, they should only perform the minimal behavior. The intent of the platform specific IMs is to alert the initiation core of the recovery condition and to also initialize enough of the platform I/O complex and memory for recovery. The modules marked “recovery dispatch” can have Interface Import Table references to non-recovery IMs. These referenced non-recovery IMs need to be stored in the boot block to insure all required modules are in a secure non-corruptible area. This implies that during a normal update an IM newly referenced by recovery code, and that is in a non-boot block area, should be migrated to the boot block.

[0007] **Figure 1** is a platform module dispatch for such a recovery process. Process 100, shown in **Figure 1** begins with operation 105 in which the core is initialized. In operations 110 and 115, all IMs marked for recovery will be executed and then the driver execution initial program loader (IPL) will be executed at operation 120.

[0008] If the boot mode is not equal to recovery, then subsequent IMs are executed at operation 125, and after each a determination is made as to whether the returned boot mode is equal to recovery. If not, no recovery is required and the driver execution IPL is executed. If a recovery is necessary, the boot mode is set to recovery and the boot mode is saved in non-volatile storage at operation 130. From there the system is reset at operation 135 and the core is initialized in recovery mode at operation 140.

BRIEF DESCRIPTION OF THE DRAWINGS

[0009] The present invention is illustrated by way of example and not intended to be limited by the figures of the accompanying drawings in which like references indicate similar elements and in which:

[0010] **Figure 1** is a platform module dispatch for such a recovery process;

[0011] **Figure 2** is a diagram illustrating an exemplary computing system 200 for implementing the present invention;

[0012] **Figures 3** is a process flow diagram in accordance with the present invention; and

[0013] **Figures 4A, 4B, and 4C** depict a firmware volume update in accordance with the present invention.

DETAILED DESCRIPTION

[0014] The method and apparatus described herein may be used for the dynamic inclusion or exclusion of initialization modules within the set of initialization modules designated recovery initialization modules. In one embodiment the BIOS system of a computing system, having a component firmware architecture (i.e., componentized BIOS), is updated through the inclusion of a new initialization module (IM). The new IM is evaluated to determine if it is designated as recovery. If so, the new module is tagged as being necessary for recovery and stored to the recovery block, a fault tolerant block within a file volume. In an alternative embodiment, all of the IMs currently designated as recovery IMs (RIMs) are evaluated to determine if a dependency upon the new IM exists. If any RIM is dependent upon the new IM, the new IM is tagged as a graduate to recovery. Under conditions of recovery restart the firmware will only execute those modules that are marked for recovery or have been graduated to recovery.

[0015] The present invention allows an IM to be designated as being required for recovery only when necessary. It does not encumber the collection of recovery IM's to be large when not required by associated dependency tables. The present invention provides the ability to make an automatic real-time determination of whether or not an IM is required for recovery.

[0016] **Figure 2** is a diagram illustrating an exemplary computing system 200 for implementing the recovery set inclusion/exclusion process of the present invention. The method of dynamic designation of initialization modules as being necessary for recovery described herein can be implemented and utilized within computing system 200, which can represent a general-purpose computer, portable computer, or other like device. The

components of computing system 200 are exemplary in which one or more components can be omitted or added. For example, one or more memory devices can be utilized for computing system 200.

[0017] Referring to **Figure. 2**, computing system 200 includes a central processing unit 202 and a signal processor 203 coupled to a display circuit 205, main memory 204, static memory 206, and mass storage device 207 via bus 201. Computing system 200 can also be coupled to a display 221, keypad input 222, cursor control 223, hard copy device 224, input/output (I/O) devices 225, and audio/speech device 226 via bus 201.

[0018] Bus 201 is a standard system bus for communicating information and signals. CPU 202 and signal processor 203 are processing units for computing system 200. CPU 202 or signal processor 203 or both can be used to process information and/or signals for computing system 200. CPU 202 includes a control unit 231, an arithmetic logic unit (ALU) 232, and several registers 233, which are used to process information and signals. Signal processor 203 can also include similar components as CPU 202.

[0019] Main memory 204 can be, e.g., a random access memory (RAM) or some other dynamic storage device, for storing information or instructions (program code), which are used by CPU 202 or signal processor 203. Main memory 204 may store temporary variables or other intermediate information during execution of instructions by CPU 202 or signal processor 203. Static memory 206 can be, e.g., a read only memory (ROM) and/or other static storage devices, for storing system firmware or other information or instructions, which can also be used by CPU 202 or signal processor 203. Mass storage device 207 can be, e.g., a hard or floppy disk drive or optical disk drive, for storing information or instructions for computing system 200.

[0020] Display 221 can be, e.g., a cathode ray tube (CRT) or liquid crystal display (LCD). Display device 221 displays information or graphics to a user. Computing system 200 can interface with display 221 via display circuit 205. Keypad input 222 is an alphanumeric input device with an analog to digital converter. Cursor control 223 can be, e.g., a mouse, a trackball, or cursor direction keys, for controlling movement of an object on display 221. Hard copy device 224 can be, e.g., a laser printer, for printing information on paper, film, or some other like medium. A number of input/output devices 225 can be coupled to computing system 200. The dynamic determination to include or exclude IMs within the set of RIMs in accordance with the present invention can be implemented by hardware and/or software contained within computing system 200. For example, CPU 202 or signal processor 203 can execute code or instructions stored in a machine-readable medium, e.g., main memory 204. The machine-readable medium may include a mechanism that provides (i.e., stores and/or transmits) information in a form readable by a machine such as computer or digital processing device. For example, a machine-readable medium may include a read only memory (ROM), random access memory (RAM), magnetic disk storage media, optical storage media, flash memory devices. The code or instructions may be represented by carrier-wave signals, infrared signals, digital signals, and by other like signals.

[0021] **Figure 3** is a process flow diagram in accordance with one embodiment of the present invention. Process 300, shown in **Figure 3** begins with operation 305 in which system firmware is updated with a new initialization module, IM_X. The firmware update utility makes an evaluation to determine if IM_X is designated as a RIM. If IM_X is designated as a RIM, then, at operation 310 all IMs that import services from IM_X are determined. These IMs communicate with IM_X, and provide services for the successful

chip silicon resources (e.g., flashpart). IMs D and E are not required for recovery and therefore are not tagged and not protected.

[0026] **Figure 4B** depicts a firmware update in which IMs A and B will be replaced. As shown in **Figure 4B**, the updated IM_A is marked as recovery, but the updated IM_B is not. For example, the component provider may simply provide the updated code and isn't concerned whether IM_B is recovery or not.

[0027] **Figure 4C** depicts the firmware volume after the dynamic designation process of the present invention. As shown in **Figure 4C**, the updated IM_A is designated as recovery. Although the updated IM_B was not designated by the provider as recovery, it has been graduated to recovery due to IM_A's dependence upon it. Where IM_C had been designated as recovery due to IM_A's dependence, IM_C has now been demoted from recovery to non-recovery status. This is due to the fact that the updated IM_A does not depend on IM_C. The sole reason IM_C had been designated as recovery was the dependence upon IM_C of the original IM_A. Likewise, IM_E, which had not been designated as recovery, may be dynamically graduated to recovery if the firmware update utility determines that a RIM (e.g., IM_A) depends upon IM_E.

[0028] This dynamic inclusion into (e.g., IM_E), and exclusion from (e.g., IM_C) the recovery set, helps to ensure that all IMs required for recovery are protected while helping to keep the number of recovery initialization modules to a minimum. This helps to ensure that a computing system in recovery mode can successfully execute all of the necessary IMs to effect a restoration of the firmware.

[0029] In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the

broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative sense rather than a restrictive sense.

0643904-03001
TOEBO"406460